Les Cahiers du GERAD ISSN: 0711–2440

A Greedy Variable Neighborhood Search Heuristic for the MaxSumSum p-Dispersion Problem

B. Saboonchi, P. Hansen,
S. Perron

G–2012–46

August 2012

A Greedy Variable Neighborhood Search Heuristic for the MaxSumSum p-Dispersion Problem

Behnaz Saboonchi
Pierre Hansen
Sylvain Perron

GERAD & HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada, H3T 2A7

behnaz.saboonchi@gerad.ca
pierre.hansen@gerad.ca
sylvain.perron@gerad.ca

August 2012

Les Cahiers du GERAD

G–2012–46

Abstract

The MaxSumSum (maximum diversity) problem consists of the selection of p facilities among n candidate locations in a way that the total sum of the distances between each pair of the located facilities is maximized. The Basic Variable Neighborhood Search heuristic (`BVNS`) has already been applied to solve this problem with success. In this work we have developed a Greedy Variable Neighborhood Search heuristic which adds a new type of plateau search mechanism to its general framework. This newly incorporated local search technique helps the exploration of the solution space and facilitates finding higher quality solutions. The proposed solution procedure further improves the already high performance of the `BVNS` and finds new improved solutions for several of the largest benchmark datasets in the literature.

Key Words: Combinatorial optimization, Dispersion problems, Metaheuristics, Variable Neighborhood Search.

Acknowledgments: This research was funded by NSERC (Natural Sciences and Engineering Research Council of Canada) grant PGSD2-392404-2010, and FRQNT (Fonds de recherche du Québec - Nature et technologies) grant 134582. Pierre Hansen has been partially supported by NSERC grant 105574-2007, Sylvain Perron has been partially supported by NSERC grant 327435-06, and they were both partially supported by FRQNT team grant PR-131365.

1 Introduction

In the family of the dispersion problems, given a set of n vertices we intend to select a subset of size p in a way that a function of the distance among the selected vertices is maximized. This is useful when some measure of diversity in the solutions is desirable. For instance in the logistics context it can be used in the location of missile silos where dispersion can reduce the chances of all of them being attacked by an attacker or for locating obnoxious facilities to be far from population zones [4]. The dispersion can also be a desirable factor when it comes to franchise location problems where we intend to avoid the cannibalization effects within the chain. The difference is not always translated into the physical distance. For instance dispersion models can also be used in order to design a portfolio of new products where it is desirable to enter the market with a group of products which are as dissimilar as possible in terms of the quality, price, shape etc. Another example would be in multi-objective problems where the decision maker may be interested in selecting a collection of solutions as far as possible for each objective [13].

Erkut and Neuman [5] propose four different types of the dispersion models based on different dispersion metrics. The first one is the MaxMinMin problem which maximizes the minimum distance between each pair of facilities. The second one is the MaxSumMin which seeks to maximize the sum of the minimum distances from each facility to its closest neighbor. The third formulation is called MaxMinSum which takes the sum of the distances from each facility to all its neighbors, and maximizes the minimum sum of the distances. Finally the fourth formulation corresponds to the MaxSumSum which aims at maximizing the sum of all the hub distances for all located facilities. This model tries to locate p facilities far from a given set of nodes and far from each other and is the one studied in this paper.

Hansen and Moon prove that the discrete version of the MaxSumSum p-dispersion problem on general networks is strongly NP-complete, by reduction to the stable set problem [2, 5, 9, 10, 14]. Yet, metaheuristics have shown to be very successful in finding high quality solutions to this problem and have been widely discussed and compared in the literature [1, 9, 10]. An extensive comparison is done by Martí et al. [10] where they compare 10 heuristics and 20 applications of metaheuristics for this problem. They conclude that the Basic Variable Neighborhood Search (BVNS) method by Brimberg et al. [2] and the Iterated Tabu Search (ITS) method by Palubeckis [12] are the most powerful applications of metaheuristics.

The ITS [12] applies a tabu search step followed by a local search in case a better incumbent solution has been found. Then a perturbation procedure is applied by swapping a random number of selected and unselected points which is similar to the shake procedure in VNS except that the shake size is random at each iteration.

The VNS method applied by Brimberg et al. [2] to the heaviest k-subgraph problem (HSP) can also be adapted to solve the MaxSumSum problem. In fact HSP is more general than the MaxSumSum problem as the edge weights do not have to represent distances and the graph is not necessarily fully connected. They compare the basic VNS (BVNS), skewed VNS (SVNS) and the greedy add and greedy drop construction procedures followed by VNS, and finally conclude that the BVNS is the overall best method. The BVNS consists of a random shaking perturbation strategy followed by a local search construction phase. Their data structure allows an efficient update of the values and thus has also been applied in this paper. The BVNS is among the most successful methods to address the MaxSumSum problem and in this work we further improve the capabilities of the VNS by developing more elaborated modules.

Another successful recent method is the Iterated Greedy metaheuristic (IG) and its fine-tuned version (TIG) by Lozano et al. [9] which generates a sequence of solutions by iterating over greedy construction and destruction phases. This method has not been considered in the comprehensive literature review of Martí et al. [10].

Finally, Wang et al. [14] compare the most successful candidate methods for the MaxSumSum problem with their Learnable Tabu Search method guided by Estimation of Distribution Algorithm (LTS-EDA). Their method can extract knowledge during the tabu search procedure and adapts the search structure. The clustered EDA is a learnable constructive method in order to create new starting solutions, coupled with the TS as an improvement method. They use either some of the executable codes provided by various authors or code the suggested algorithms in the literature themselves, and compare all the metaheuristics including the ITS, VNS, TIG and LTS-EDA under the same conditions. Their final comparison shows that the LTS-EDA obtains the best improvements over the existing large benchmark test problems. The best results are obtained in the long run version of the experiments, that will be used as the best ever solutions in the literature in Section 4.

In Section 2 we describe the problem in graph theoretical terms followed by its mixed integer formulation. Section 3 presents a detailed explanation of our proposed greedy `VNS` heuristic solution procedure for the MaxSumSum p-dispersion problem. Then we discuss our computational experiments on the largest known benchmark test problems and finally conclude the paper by highlighting our contributions and suggestions for future research.

2 Problem statement and mathematical formulation

This section expresses the MaxSumSum p-dispersion problem in graph theoretical terms and then presents its respective mathematical formulation. Let $V = \{v_i, \forall i = 1, \dots, n\}$, be a set of n vertices (potential locations) and v_i representing each member of this set. Let E be the set of $\binom{n}{2}$ edges of an undirected and fully connected graph $G(V, E)$, with $d_e \geq 0$ representing the distance over each edge $e \in E$. The value p is an integer such that $3 \leq p \leq |V|$. We define S as any subset of p vertices such that $S \subseteq V, |S| = p$. The subset of the vertices not present in the current solution is defined as $\bar{S}$ such that $\bar{S} = V \setminus S, |\bar{S}| = n - p$.

The objective function value $f(S)$ is defined as the total sum of the distances among all the p selected vertices induced by the subset S:

$$f(S) = \sum_{v_i \in S} \sum_{v_j \in S} d(v_i, v_j).$$

The MaxSumSum p-dispersion problem intends to find the optimal subgraph $G(S^*, E(S^*))$, where:

$$S^* = \arg\max_S f(S).$$

The MaxSumSum p-dispersion problem could be modeled as the following 0-1 mixed integer program as suggested in [5]:

$$\begin{array}{rll}
\max & \sum_{i=1}^{n} Z_i & \\
s.t. & Z_i \leq M x_i & 1 \leq i \leq n \\
& Z_i \leq \sum_{j=1}^{n} d(v_i, v_j) x_j & 1 \leq i \leq n \\
& \sum_{i=1}^{n} x_i = p & \\
& x_i = \{0, 1\} & 1 \leq i \leq n,
\end{array}$$

where x_i is a binary decision variable defining if vertex v_i is selected, $d(v_i, v_j)$ is the distance between any pair of the located facilities at locations i and j and M is a sufficiently large number which could be set as the sum of the p largest distances among the n vertices. The distances between all the vertices are taken as an input and stored in an $n \times n$ upper-triangular matrix with $d(v_i, v_i) = 0$.

3 VNS for the p-dispersion-sum problem

Variable Neighborhood Search (`VNS`) is a metaheuristic or framework for building heuristics which is based on the idea of a systematic change of the neighborhood in order to escape from the valleys surrounding local optima, followed by a local search to find improved solutions. This general method has been proposed by

Mladenović and Hansen [11] and has proven to lead to very successful heuristics for solving large combinatorial programs with applications in location theory, cluster analysis and several other fields. For a recent survey of the theoretical developments and applications including several hundred references see [7, 8].

The Basic Variable Neighborhood Search method (BVNS) has already been applied to the heaviest k-subgraph problem and has shown to be among the most efficient methods compared to other heuristics for the maximum diversity problem [2, 10]. Brimberg et al. [2] suggest the examination of different VNS strategies and also extensive experimental testings of VNS on different types of graphs as a future extension to their work. Therefore, we have decided to develop a more elaborated heuristic method within the VNS framework that is well-suited to the MaxSumSum p-dispersion problem.

In order to represent the solution at each step of the heuristic we use the data structure suggested in [2]. The solution is represented by an array of the n indices corresponding to each vertex or candidate location, where the first p elements correspond to the subset of the current solution S.

The solution space U is represented by the $\binom{n}{p}$ subsets of V with cardinality p. In order to apply VNS a metric function is defined to evaluate the distance between any two solutions S and S':

$$\delta(S, S') = \delta(S', S) = |S \setminus S'|,$$

i.e, the Hamming distance between the indicator vectors.

Based on the metric distance function defined above, the neighborhood of size k of a solution S is defined as:

$$N_k(S) = \{S' \in U | \delta(S, S') = k\}; k = 1, 2, \ldots, \min\{p, n-p\}.$$

Throughout this paper the following notations are used:

- $S_{best/cur}$: the best/current solution set corresponding to the best/current objective function value;
- $f(S_{best/cur})$: the best/current objective function value that corresponds to the sum of the distances among all the selected vertices in the best/current solution set S;
- $W(v_i)$: the sum of the distances from any vertex v_i $(i = 1, \ldots, n)$ to all the vertices in the solution set S;
- v_{exit}: the vertex inside the solution set that is a candidate to *leave* the solution set ($v_{exit} \in S$);
- v_{enter}: the vertex outside the solution set that is a candidate to *enter* the solution set ($v_{enter} \in \bar{S}$).

These values are first computed at the construction of the initial solution and are updated each time a new solution is found.

In Algorithm 1 we define our VNS function and then in the following sections we explain in details the functions embedded in our general framework. The stopping criterion is the total execution time t_{max} and the already elapsed cumulative time in the overall procedure is noted by $t_{elapsed}$. The k_{min} and k_{step} (step size) parameters are set by default to 1, and the k_{max} (maximum shake size) is set to a coefficient of $\min\{p, n-p\}$ as discussed later.

3.1 Initialization

The initial solution could be created at random or in a greedy manner. Based on the *random* method the initial solution is simply created by choosing p vertices at random.

Two *Greedy* construction heuristics have been widely used in the literature in order to create initial solutions for the dispersion problems [6]. The *Greedy deletion* heuristic starts with all the n vertices and eliminates one vertex at each iteration. For this problem the deletion candidate is the one with the smallest sum of the distances value with the rest of the remaining vertices at each iteration, and the ties are broken arbitrarily. Of course this procedure is repeated $(n-p)$ times until exactly p vertices remain in the solution set.

```
function VNS ($k_{min}, k_{step}, k_{max}, S$);
$S_{best} \leftarrow$ Initialize($S$);
$S_{cur} \leftarrow S_{best}$;
$t_{elapsed} = 0$;
$k_{max} = \min\{p, n-p\}$;
while $t_{elapsed} \leq t_{max}$ do
    $k_{cur} \leftarrow k_{min}$;
    while $k_{cur} \leq k_{max}$ and $t_{elapsed} \leq t_{max}$ do
        $S_{cur} \leftarrow$ Shake($S_{cur}$) ;
        $S_{cur} \leftarrow$ LocalSearch($S_{cur}$) ;
        $S_{cur} \leftarrow$ RefinedLocalSearch($S_{cur}$) ;
        if $f(S_{cur}) \geq f(S_{best})$ then
            $S_{best} \leftarrow S_{cur}$;
            $k_{cur} \leftarrow k_{min}$;
        else
            $k_{cur} \leftarrow k_{cur} + k_{step}$;
        end
    end
end
```

Algorithm 1: Pseudo code for the VNS framework

The *Greedy add* heuristic selects a starting vertex at random and creates the complete solution set in $(p-1)$ iterations. As a result, the size of the under construction solution set is smaller than p and will gradually reach the complete size as the construction phase is executed. If we represent the solution set under construction (i.e. the size of the set is smaller than p) as C, and the set of the vertices outside this set as $\bar{C}$, the entering candidate would be the one with the largest sum of distances value with the vertices $v_i \in C$ [1, 9]. At each iteration the objective function value is the total sum of the $W(v_i)$ values for all the vertices $v_i \in C$. After the addition of the entering vertex $v_{enter} \in \bar{C}$, the existing sum of the distances values will be updated as: $W(v_i) + d(v_i, v_{enter})$ for all the $v_i \in C$. As the result the objective function value after the addition of each v_{enter} will be: $f(s_{cur}) + \sum_{v_i \in C} d(v_i, v_{enter})$.

This heuristic could be repeated n times based on different starting vertices and then the best one leading to the highest objective value could be selected. Based on preliminary results we know that this heuristic is very time consuming (much more than the *Greedy deletion* heuristic), and for large datasets it is not worthwhile to take this procedure just to further improve the initial solution. In order to overcome this drawback the *Greedy add* heuristic is initialized by choosing the two furthest vertices as the initial vertices and by repeating the above-mentioned procedure $(p-2)$ times. We observed empirically that the results obtained by this method are among the highest possibilities for the *Greedy add* heuristic without spending too much computational time on the initial solution.

3.2 Local search

After having created the initial solution, the `LocalSearch` procedure is implemented performing 1-interchange swaps on the current solution as shown in Algorithm 2. This means that at each iteration only *one* vertex is swapped at a time. The swap could be done whenever the first (first improvement strategy) or the best (best improvement strategy) contribution is made to the current objective value.

In order to start the `LocalSearch` procedure the gain obtained from swapping the selected entering candidate with the selected leaving candidate should be evaluated. The two main `Contribution` and `Update` functions will be explained in details in the following subsection.

```
function LocalSearch($S$);
$gain = 0$;
while $gain \geq 0$ do
    $(v_{exit}, v_{enter}, gain) \leftarrow$ Contribution($S$) ;
    if $gain > 0$ then
        Swap($v_{exit}, v_{enter}$);
        Update($v_{exit}, v_{enter}, S$);
    end
end
```

Algorithm 2: Pseudo code for the Local Search

3.2.1 Contribution and Update

In the proposed `LocalSearch` procedure the `Contribution` function can determine the first or best entering candidate $v_{enter} \in \bar{S}$, as well as its corresponding contribution to the current objective function value.

In order to initiate the `Contribution` function a random leaving candidate $v_{exit} \in S$, and a random entering candidate $v_{enter} \in \bar{S}$ are chosen. The change in the objective function value resulted from any swap will be calculated as :

$$change \leftarrow W(v_{enter}) - W(v_{exit}) - d(v_{exit}, v_{enter}).$$

With the first improvement strategy the `Contribution` function stops as soon as an improving solution is found, as the result in the worst case it is implemented in $O(p(n-p)) = O(np)$ time per `LocalSearch` iteration. Of course the best improvement strategy will be implemented in exactly $O(np)$ time as every possible swap should be evaluated.

The `Update` procedure performs all the required updates before proceeding to the subsequent iteration. It is implemented in two different phases in order to update all the $W(v_i)$ sum of the distances values, and then to update the objective function value $f(S_{cur})$. The update for the $W(v_i)$ is straightforward and is performed in $O(n)$ total time, whereas the update of the objective function is done in $O(p)$ time at each iteration. Thus the updated values would be:

$$W(v_i) = W(v_i) + d(v_i, v_{enter}) - d(v_i, v_{exit}), i = 1, 2, \ldots, n;$$

$$f(S_{cur}) = f(S_{cur}) + \sum_{i=1}^{p} d(v_i, v_{enter}) - \sum_{i=1}^{p} d(v_i, v_{exit}).$$

3.3 Refined local search

The existence of plateaus or the flat landscapes of the MaxSumSum problem make it more difficult for the descent procedures to find an improvement in the objective function especially as the size and the density of the graphs increase [2]. This has inspired us to develop a plateau search mechanism within the `VNS` metaheuristic framework.

The `RefinedLocalSearch` procedure demonstrated in Algorithm 3 is efficient both in the presence and the absence of plateaus. The reason is that it makes all the possible swaps that will improve the quality of the solution set by exchanging the low quality vertices with the ones of higher quality which will facilitate further improvements by the `LocalSearch` procedure in the subsequent iterations. It should be noted that in Algorithm 3 the smallest $W(v_{exit}) \in S$ is found in $O(p)$ time and the largest $W(v_{enter}) \in \bar{S}$ is determined in $O(n-p)$ time.

```
function RefinedLocalSearch($S$);
$gain$ = 0;
while $gain \geq 0$ do
    Find $v_{exit}$ in the solution subset $S$ with the smallest $W(v_{exit})$;
    Find $v_{enter}$ outside the solution subset $\bar{S}$ with the largest $W(v_{enter})$;
    $change \leftarrow W(v_{enter}) - W(v_{exit}) - d(v_{exit}, v_{enter})$ ;
    if $change > 0$ then
        Swap($v_{exit}, v_{enter}$);
        Update($v_{exit}, v_{enter}, S$);

end
```

Algorithm 3: Pseudo code for the refined local search

Of course the `RefinedLocalSearch` procedure never worsens the actual solution and could even lead to improvements in the objective function value if the whole plateau is removed. As discussed later in Section 4, the addition of this module improves significantly the quality of the obtained solutions.

3.4 Shake

The perturbation in most `VNS`-based heuristics is done in a simple manner by choosing a random vertex from the k_{th} neighborhood, i.e. $N_k(S)$ from the current solution S and then repeating k times the random swap move. The `RandomShake` function does so by choosing one random leaving and entering candidate at each iteration with updates in between each swap. However, here we have developed two additional shake functions in order to control the perturbation operation in a more intelligent manner.

The `SemiGreedyShake` function fixes a random leaving candidate from the current solution set S and then chooses an entering candidate that has the highest sum of the distances value in case it replaces the exit candidate, i.e. the highest $W(v_{enter}) - d(v_{exit}, v_{enter})$ value. As the result this shake method does not guarantee that a deterioration in the objective function value would not occur, yet it simply chooses a reasonable entering candidate after having fixed the leaving candidate. This procedure is repeated until the shake size of k is attained. Each iteration is performed in $O(n-p)$ time and after each swap the `Update` function is called.

In order to have a more intensified shake operation the `GreedyShake` function has been developed which for a shake of size k, selects the k vertices with the smallest $W(v_i)$ values for all $v_i \in S$, and swaps all of them with the k vertices with the largest $W(v_i)$ values for all $v_i \in \bar{S}$. This greedy fashion of selecting the entering candidates could provide better starting vertices for the subsequent `LocalSearch` procedure. In order to keep the random nature of the shake procedure, the k leaving and entering candidates are chosen all at once and are not changed while the updates are performed between the swaps. The performance of the two shake functions will be compared in details in Section 4.

4 Computational experiments

In this section we have selected four of the largest benchmark instances that were used for comparison purposes in the maximum diversity problem literature leading to 50 instances in total [2, 9, 10, 12, 14]. A brief description of the characteristics of the datasets is given below:

- MDG-a: this dataset consists of 20 matrices with real numbers randomly selected between 0 and 10 from a uniform distribution by Duarte and Martí [3] with $n = 2000$ and $p = 200$.
- MDG-c: this dataset consists of 20 matrices with $n = 3000$ and $p = 300$ (MDG-c-1 to 5), 400 (MDG-c-6 to 10), 500 (MDG-c-11 to 15) and 600 (MDG-c-16 to 20) [3].

- p5000 and p3000: these datasets consist of 10 matrices each with integer numbers generated from 0 to 100 from a uniform distribution by Palubeckis [12]. There are five instances with $n = 3000$ and $p = 0.5n$, and five instances with n =5000 and $p = 0.5n$ with a matrix density of 10, 30, 50, 80 and 100%.

First we describe our experiments that were designed to study the performance of different settings within the `VNS` framework and then we compare and analyze the tradeoffs and overall results obtained by different methods over all the test problems. Finally we tune our framework based on the preliminary results and compare our results with that of the state of the art heuristics in the literature.

All the heuristics were coded in C ++ and run on a linux machine, with 2.667 GHz and 3Gb Ram. The best results obtained after two hours of running time are reported as suggested in [10].

4.1 Preliminary experiments setup

As mentioned in Section 3 our `VNS` implementation allows various settings and methods within its framework. In order to initialize the `VNS` three different methods have been discussed: Random add (RA), Greedy add (GA) and Greedy deletion (GD). There are also three different shaking possibilities: Random shake (RS), Semi-Greedy shake (SG) and Greedy shake (GS). In the general framework presented in Section 3 the shake size at each iteration increases systematically and will be reset to k_{min} whenever an improvement is made or when the k_{max} value is reached. The k_{max} is a parameter whose value by default is $\min\{p, n-p\}$, which could be a large value depending on the problem size. As the result we are interested in trying smaller values, i.e. $0.5 * \min\{p, n-p\}$ and $0.75 * \min\{p, n-p\}$ as well to verify if it helps improve the performance of the heuristic. Besides each experiment could be done with or without the `RefinedLocalSearch` module. This will lead to $3 \times 3 \times 3 \times 2 = 54$ combinations of different `VNS` modules.

On the other hand at each iteration of `VNS` either an improvement is made or not. In case of no improvement a decision on how to start the next iteration should be made. The next iteration is either started form the already best solution obtained (from best or FB), or from the current solution just obtained (from current or FC). The former will lead to more intensification in the search, whereas the latter favors diversification. Besides, the `LocalSearch` procedure can pursue a first improvement strategy ($FirstI$) favoring more diversification, versus best improvement strategy ($BestI$) leading to more intensification. The above mentioned intensification and diversification strategies will lead to four general `VNS` frameworks. As the result the datasets are run under the $54 \times 4 = 216$ total combinations.

We do not allow longer running times in order to get further improvements, as the result the tests are run *only once* under *two hours* of running time as used in [10]. Throughout the paper we present the average *% deviation* from the best solutions obtained by the 216 combinations for each group of dataset:

$$\% \text{ deviation} = \frac{\text{best obtained value - actual value}}{\text{best obtained value}} \times 100.$$

Table 1 represents the average deviation from the best solutions obtained for all the 50 data instances for each of the *four* `VNS` general frameworks, *three* initialization methods, *three* shaking strategies, *three* possible k_{max} sizes and the presence or absence of the `RefinedLocalSearch` module. The average deviations are all calculated based on the results presented under the G-VNS column of Tables 2 to 4 which refer to the best solutions obtained by a *single* run of the 216 combinations. The smallest average deviation values for each group are shown in bold under the *Average* column of Table 1. Same representation has also been done to highlight the smallest average deviation values for each of the four datasets.

The first part of Table 1 which refers to the four general `VNS` frameworks reveals that the current iteration strategy and first contribution local search strategy *(FC-FirstI)* leads to the overall lowest average deviation. This is also true for each individual dataset except for the p3000 instances where the *(FB-FirstI)* strategy leads to the lowest average deviation. In the second part which refers to the problem initialization methods, the Greedy deletion strategy (GD) consistently leads to the lowest average deviation for all datasets. In the third group which addresses the shaking strategy the Greedy shake (GS) strategy has a significantly lower average deviation across all the datasets which emphasizes the importance of more intelligent shaking

Table 1: Average % deviation for individual methods

	MDG-a	MDG-c	p3000	p5000	Average
General framework					
FC-FirstI	**0.72**	**0.45**	0.24	**0.19**	**0.4**
FC-BestI	0.99	0.61	0.38	0.31	0.57
FB-FirstI	0.81	0.5	**0.22**	**0.19**	0.43
FB-BestI	1	0.64	0.32	0.27	0.56
Initialization					
RA	5.57	0.96	0.66	0.56	1.94
GA	2.85	0.42	0.14	0.11	0.88
GD	**2**	**0.29**	**0.07**	**0.05**	**0.6**
Shake method					
RS	6.34	1.07	0.74	0.62	2.19
SG	2.49	0.35	0.07	0.06	0.74
GS	**1.73**	**0.24**	**0.06**	**0.04**	**0.52**
Shake max size					
0.5p	**3.41**	**0.54**	**0.28**	**0.24**	**1.12**
0.75p	3.51	0.56	0.29	**0.24**	1.15
p	3.63	0.57	0.3	**0.24**	1.19
Refined local search					
Yes	**1.25**	**0.19**	**0.03**	**0.03**	**0.38**
No	5.79	0.92	0.55	0.45	1.93

strategies compared to pure random ones. The only parameter that we changed in our experiments is the maximum shake size which is presented in the last comparison group. As it is seen the smaller maximum shake size is slightly better for all instances, yet the difference among them is not significant. Finally in the last part it is clearly observed that the existence of the `RefinedLocalSearch` module leads to significantly lower average deviations which highlights the importance of addition of this new local search mechanism to the classical body of `VNS`.

To summarize, four important observations can be highlighted:

- The current iteration strategies (*FC-FirstI* and *FB-FirstI*) lead to the lowest average deviations compared to other general frameworks.
- The Greedy deletion initialization method leads to the highest average quality.
- The Greedy Shake (GS) method leads to the lowest average deviations over all data instances.
- The Refined Local Search plateau search mechanism contributes strongly to the creation of high quality solutions.

4.2 Results and analysis

The best preliminary results obtained in Section 4.1 are presented under the greedy `VNS` (G-VNS) column in Tables 2 to 4. Based on the findings in the previous section, the following `VNS` setting is selected in order to do the final experiments:

- Each iteration will start form the current solution just obtained (the current iteration strategie), and the Local Search function will randomly choose between the first or best improvement strategies.
- The Random add initialization method is selected in order to obtain more diversified solutions due to its random starts. The Greedy deletion strategy leads to robust hight quality solutions, yet obtains the same results in multiple runs based on our preliminary experiments.
- The Greedy shake method is used with the maximum shake size randomly selected as $0.5 * \min\{p, n-p\}$ or $0.75 * \min\{p, n-p\}$.
- The Refined Local Search plateau search mechanism will be used in all the experiments.

All the instances are run 15 times with the two hours of running time constraint under the above selected settings and the obtained solutions for all the 50 data instances are presented under the TG-VNS columns of Tables 2 to 4.

Table 2: Comparison of the best known results for the MDG-a instances

Instance	n	p	Best known	G-VNS	TG-VNS		
					Best	Average	CV
MDG-a-21	2000	200	114271 (ITS)	**0**	**0**	3.33	0.00003
MDG-a-22	2000	200	114327 (ITS)	**0**	**0**	**0**	**0**
MDG-a-23	2000	200	114195 (ITS)	**0**	**0**	**0**	**0**
MDG-a-24	2000	200	114093 (ITS)	**0**	**0**	5.2	0.00002
MDG-a-25	2000	200	114196 (ITS)	**0**	**0**	**0**	**0**
MDG-a-26	2000	200	114265 (ITS)	**0**	**0**	**0**	**0**
MDG-a-27	2000	200	114361 (ITS)	**0**	**0**	**0**	**0**
MDG-a-28	2000	200	114327 (ITS)	**0**	**0**	**0**	**0**
MDG-a-29	2000	200	114199 (ITS)	**0**	**0**	1.87	0.000005
MDG-a-30	2000	200	114229 (ITS)	**0**	**0**	**0**	**0**
MDG-a-31	2000	200	114214 (ITS)	**0**	**0**	6.6	0.00009
MDG-a-32	2000	200	114214 (ITS)	**0**	**0**	7	0.00007
MDG-a-33	2000	200	114233 (ITS)	**0**	**0**	3.73	0.00002
MDG-a-34	2000	200	114216 (ITS)	**0**	**0**	**0**	**0**
MDG-a-35	2000	200	114240 (ITS)	**0**	**0**	0.87	0.000003
MDG-a-36	2000	200	114335 (ITS)	**0**	**0**	**0**	**0**
MDG-a-37	2000	200	114255 (ITS)	**0**	**0**	**0**	**0**
MDG-a-38	2000	200	114408 (ITS)	**0**	**0**	1.73	0.00001
MDG-a-39	2000	200	114201 (ITS)	**0**	**0**	**0**	**0**
MDG-a-40	2000	200	114349 (ITS)	**0**	**0**	**0**	**0**

Table 3: Comparison of the best known results for the MDG-c instances

Instance	n	p	Best known	G-VNS	TG-VNS		
					Best	Average	CV
MDG-c-1	3000	300	24924685 (BVNS)	**0**	**-1659**	270.33	0.003
MDG-c-2	3000	300	24909199 (BVNS)	**-3347**	**-3347**	**-2069.13**	0.005
MDG-c-3	3000	300	24900820 (ITS)	**-4398**	**-4398**	**-2266.93**	0.011
MDG-c-4	3000	300	24904964 (BVNS)	**-4746**	**-4746**	**-916.4**	0.006
MDG-c-5	3000	300	24899703 (ITS)	3999	3999	5139.4	0.005
MDG-c-6	3000	400	43465087 (ITS)	20139	20139	24533.47	0.008
MDG-c-7	3000	400	43477267 (BVNS)	**0**	**0**	737.13	0.002
MDG-c-8	3000	400	43458007 (BVNS)	**-7565**	**-7565**	**-2944.33**	0.005
MDG-c-9	3000	400	43448137 (BVNS)	**0**	**0**	394.87	0.001
MDG-c-10	3000	400	43476251 (ITS)	10690	10690	10782.73	0.001
MDG-c-11	3000	500	67009114 (BVNS)	**-12018**	**-12018**	**-6720.4**	0.006
MDG-c-12	3000	500	67021888 (ITS)	7718	7718	9397.87	0.006
MDG-c-13	3000	500	67024373 (BVNS)	**0**	**0**	392	0.001
MDG-c-14	3000	500	67024804 (BVNS)	**-5386**	**-5386**	**-3395.87**	0.004
MDG-c-15	3000	500	67056334 (BVNS)	1624	**0**	2448.2	0.004
MDG-c-16	3000	600	95637733 (BVNS)	**-1196**	**-1196**	174.53	0.001
MDG-c-17	3000	600	95645826 (ITS)	74713	74713	76052	0.002
MDG-c-18	3000	600	95629207 (ITS)	97066	97100	100356.2	0.005
MDG-c-19	3000	600	95633549 (ITS)	34385	34385	35069.8	0.001
MDG-c-20	3000	600	95643586 (ITS)	59104	59104	59509.07	0.001

Table 4: Comparison of the best known results for the p3000 and p5000 instances

Instance	n	p	Best known	G-VNS	TG-VNS		
					Best	Average	CV
p3000-1	3000	1500	6502308 (LTS)	208	**-22**	422.13	0.01
p3000-2	3000	1500	18272568 (LTS)	354	**0**	362.93	0.001
p3000-3	3000	1500	29867138 (ITS)	**0**	**0**	634.8	0.002
p3000-4	3000	1500	46915044 (LTS)	96	86	429.33	0.001
p3000-5	3000	1500	58095426 (LTS)	237	**-41**	1101.4	0.002
p5000-1	5000	2500	17509215 (LTS)	966	**-112**	419.93	0.005
p5000-2	5000	2500	50102729 (LTS)	1196	**-156**	836.6	0.004
p5000-3	5000	2500	82039686 (LTS)	1482	426	2746.73	0.003
p5000-4	5000	2500	129413112 (LTS)	59	**-112**	1878.13	0.001
p5000-5	5000	2500	160597781 (LTS)	430	**-15**	843.67	0.001

In the *Best Known* column of Tables 2 to 4 the best known solutions for each of the instances are presented which have been found by different powerful heuristic methods from various references. It should be noted that the best results in the literature have been obtained by longer running times of 5h for smaller instances and 10h for larger instances. Despite the fact that our running time is only two hours, we still keep these best solutions as a benchmark in order to verify the quality of our solutions.

The *G-VNS* column represents the proposed Greedy `VNS` results which shows the difference between the best known solutions in the literature and the best solutions obtained in the preliminary experiments of 216 `VNS` module combinations presented in Section 4.1. A negative value represents an improvement in the best solutions ever reported, a value of zero means that we have equaled the best ever value and a positive value means that our solution is lower than the one reported in the literature. Here no average results are presented as in the preliminary experiments each combination is run only once.

The following three columns are called *TG-VNS* representing the difference between the best known literature results and the *Best* and *Average* results obtained by multiple runs of the Tuned Greedy `VNS` setting presented in Section 4.2. The reported *Best* values are the best among 15 runs of the tuned greedy heuristic, yet in order to have a better idea of the overall quality of all the 15 runs, the coefficient of variation *(CV)* is presented for each dataset under the *CV* column:

$$\text{CV} \;=\frac{\text{standard deviation}}{\text{mean}}\times 100.$$

As it is seen for all the 50 instances in Tables 2 to 4 the coefficient of variation value is very small, much less than 0.01%, which shows that our random methods are also very robust.

As illustrated by the bold values in Table 2, our proposed method finds *all* of the best ever solutions for the MDG-a instances in the G-VNS preliminary experiments and the TG-VNS *Best* case, and also 12 out of 20 in the TG-VNS *Average* case. The best solutions have been initially obtained by the Iterated Tabu Search (ITS) method in [12] and have also been found in the long run experiments of Tuned Iterated Greedy method (TIG) in [9] and the Learnable Tabu Search method (LTS-EDA) in [14]. Yet, none of these methods finds *all* the best solutions at once even in longer running times.

Most of the best results for the MDG-c instances in the literature have been obtained by the `BVNS` [2] and the ITS [12]. Our method finds equal or better new solutions for 11 instances in the G-VNS preliminary experiments, 12 instances in the TG-VNS *Best* case and 6 instances in TG-VNS *Average* case which are represented in Table 3.

The best results in the literature for the p3000 and p5000 instances represent the best ever results obtained by state of the art heuristics such as ITS, VNS, TIG and LTS-EDA under 5h and 10h running times. As shown in Table 4, the TG-VNS method finds one equal solution with the G-VNS, and 8 out of 10 equal or new solutions with the TG-VNS *Best* case, representing the best solutions in 15 runs of two hours of computation each.

5 Conclusions and future work

In this work we applied an elaborated greedy `VNS` framework over 50 of the largest data instances for the MaxSumSum p-dispersion problem. We first explained a detailed preliminary experimental setting which captured a vast number of possibilities within the `VNS` framework, and then selected the most promising setting in order to conduct the tuned experiments. We also incorporated a new local search module within the `VNS` framework coupled with elaborated shake function. Our extensive computational experiments on the largest benchmarks in the literature found new best solutions for several problems that proves the high quality of our methods.

Of course in order to have a more precise comparison of various heuristics, the same executable codes of the different authors should be run under the same conditions. Yet, we believe that the reported results represent a fair comparison with other state of the art heuristics, as we have considered the best ever solutions obtained by all the heuristic methods ever studied in the literature under long running times as our benchmark.

One of the most interesting advantages of the Variable Neighborhood Search metaheuristic is its flexibility and how it allows the decision maker to define and adapt the framework to its own problem specifications. The choice of the best setting is always a matter of time and available computational resources, and also the fact that if one is interested in a heuristic that provides more robust and higher quality solutions on average, or a method that gives the opportunity of obtaining new improved solutions over repeated runs. As the future work we suggest developing decomposed `VNS` frameworks in order to tackle large problem instances in a more reasonable time and to possibly further improve the obtained results.

References

[1] Aringhieri, R., Cordone, R. Comparing local search metaheuristics for the maximum diversity problem. The Journal of the Operational Research Society, 2011; 62(2):266–280.

[2] Brimberg, J., Mladenović, N., Urošević, D., Ngai, E. Variable neighborhood search for the heaviest k-subgraph. Computers & Operations Research, 2009; 36(11):2885–2891.

[3] Duarte, A., Martí, R. Tabu search and grasp for the maximum diversity problem. European Journal of Operational Research, 2007; 178(1):71–84.

[4] Erkut, E. The discrete p-dispersion problem. European Journal of Operational Research, 1990; 46(1):48–60.

[5] Erkut, E., Neuman, S. Comparison of four models for dispersing facilities. INFOR, 1991; 29(2):68–86.

[6] Erkut, E., Ülküsal, Y., Yeniçerioğlu, O. A comparison of p-dispersion heuristics. Computers & Operations Research, 1994; 21(10):1103–1113.

[7] Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M. Variable neighborhood search. In: Gendreau, M., Potvin, J.Y., editors. Handbook of Metaheuristics. Springer US; volume 146 of *International Series in Operations Research & Management Science*, 2010a, p. 61–86.

[8] Hansen, P., Mladenović, N., MorenoPérez, J. Variable neighbourhood search: methods andapplications. Annals of Operations Research, 2010b; 175:367–407.

[9] Lozano, M., Molina, D., García-Martínez, C. Iterated greedy for the maximum diversity problem. European Journal of Operational Research, 2011; 214(1):31–38.

[10] Martí, R., Gallego, M., Duarte, A., Pardo, E. Heuristics and metaheuristics for the maximum diversity problem. Journal of Heuristics, 2011; 1–25.

[11] Mladenović, N., Hansen, P. Variable neighborhood search. Computers & Operations Research, 1997; 24(11): 1097–1100.

[12] Palubeckis, G. Iterated tabu search for the maximum diversity problem. Applied Mathematics and Computation, 2007; 189(1):371–383.

[13] Ravi, S.S., Rosenkrantz, D.J., Tayi, G.K. Heuristic and special case algorithms for dispersion problems. Operations Research, 1994; 42(2):299–310.

[14] Wang, J., Zhou, Y., Cai, Y., Yin, J. Learnable tabu search guided by estimation of distribution for maximum diversity problems. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2012; 16:711–728.